2.2 Problem Solving and Programming: 1-hour Pack

OCR H446 · Calculators not allowed · about 60 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A sorted array holds 8 integers with indices 1 to 8:

list = [3, 8, 15, 19, 24, 31, 42, 50]

The function below performs a binary search for a target value:

```
FUNCTION BinarySearch(list : ARRAY[1:8] OF INTEGER, target : INTEGER) RETURNS
    lower <- 1
    upper <- 8
    WHILE lower <= upper
        mid <- (lower + upper) DIV 2
        IF list[mid] = target THEN
            RETURN mid
        ELSE IF list[mid] < target THEN
            lower <- mid + 1
        ELSE
            upper <- mid - 1
        ENDIF
    ENDWHILE
    RETURN -1
ENDFUNCTION
```

BinarySearch(list, 24) is called.

(a) Complete a trace table showing the values of lower, upper, mid and list[mid] for each iteration of the WHILE loop, until the target 24 is found. **(5)**

(b) State the number of comparisons with list[mid] needed to locate the value 24. **(1)**

(c) Explain why binary search cannot reliably be used on the unsorted array [8, 3, 19, 15, 24, 31, 42, 50]. **(2)**

© Revision Library | revisionlibrary.co.uk

(d) State the time complexity of binary search in Big O notation, and explain why it is more efficient than linear search, O(n), for very large data sets. **(3)**

(e) Write pseudocode for a function LinearSearch(list : ARRAY[1:8] OF INTEGER, target : INTEGER) RETURNS INTEGER that returns the index of target in list, or -1 if it is not present. **(3)**

(Total for Question 1 is 14 marks)

---

2 The array [8, 3, 6, 1] is to be sorted into ascending order using a bubble sort.

(a) Complete the trace table below, showing the state of the array after each full pass of a standard bubble sort (comparing and swapping adjacent elements left to right), and the number of swaps made in that pass. **(4)**

Start: [8, 3, 6, 1]
After pass 1: [___] , swaps = ___
After pass 2: [___] , swaps = ___
After pass 3: [___] , swaps = ___
After pass 4: [___] , swaps = ___

(b) State how a bubble sort can be made more efficient by detecting that the array is already sorted, and explain how this is recognised. **(2)**

(c) Write a procedure BubbleSort(BYREF arr : ARRAY[1:n] OF INTEGER, n : INTEGER) that sorts arr into ascending order, including the early-exit optimisation from part (b). **(6)**

(d) State the worst-case time complexity of bubble sort in Big O notation. **(1)**

(Total for Question 2 is 13 marks)

© Revision Library | revisionlibrary.co.uk

3 A stack is implemented using an array stack[1:5] with an integer pointer top, initially 0 (empty stack).

(a) State the names of the two operations used to add data to, and remove data from, a stack. **(2)**

(b) The following sequence of operations is carried out on the empty stack: **(5)**
Push(5), Push(9), Pop(), Push(3), Push(7), Pop()
Complete a trace table showing the value of top and the contents of the stack after each operation.

(c) Explain why a stack, rather than a queue, is an appropriate data structure for checking that brackets in an expression such as ((a+b)*(c-d)) are balanced. **(3)**

(d) State one real-world application of a queue in a computer system. **(1)**

(Total for Question 3 is 11 marks)

---

4 The following recursive function is defined:

```
FUNCTION Mystery(n : INTEGER) RETURNS INTEGER
    IF n = 1 THEN
        RETURN 1
    ELSE
        RETURN n * Mystery(n - 1)
    ENDIF
ENDFUNCTION
```

(a) State what Mystery(n) calculates. **(1)**

(b) Complete a trace of the recursive calls made, and the value returned at each stage, for the call Mystery(4). **(5)**

(c) Explain, using the term "base case", why calling Mystery(0) would cause a runtime error, and state the name commonly given to this type of error. **(3)**

(d) Rewrite Mystery as an iterative (non-recursive) function, MysteryIterative, using a FOR loop, that calculates the same result for positive integer values of n. **(4)**

(Total for Question 4 is 13 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.2 Problem Solving and Programming: 1-hour Pack

Question 1

(a) iteration 1: lower=1, upper=8, mid=4, list[4]=19; 19<24 so lower becomes 5 **B1**

(a) iteration 2: lower=5, upper=8, mid=6, list[6]=31; 31>24 so upper becomes 5 **B1**

(a) iteration 3: lower=5, upper=5, mid=5, list[5]=24; match found **B1**

(a) correct comparison/decision applied at each step (equal / less than / greater than target) **B1**

(a) final index returned = 5, cao **B1**

(a) Answer: Index 5 (list[5] = 24)

(b) 3, cao **B1**

(b) Answer: 3

(c) binary search relies on the data being sorted so that comparing the middle value with the target tells you which half to discard, oe **B1**

(c) on unsorted data this less-than/greater-than comparison is not reliable, so a whole half of the array could be discarded even though it contains the target, oe **B1**

(c) Answer: Binary search assumes ordering to eliminate half the search space each time; on unsorted data that assumption fails and the target could be discarded incorrectly.

(d) O(log n) **B1**

(d) explains that binary search halves the search space at each step, whereas linear search may check every item, oe **B1**

(d) as n grows very large, log n grows far more slowly than n, so binary search needs far fewer comparisons in the worst case, oe **B1**

(d) Answer: O(log n); it is faster than O(n) linear search for large n because each comparison eliminates half the remaining data rather than checking one item at a time.

(e) loop through each index of the array, e.g. FOR i <- 1 TO 8 **M1**

(e) compares list[i] to target and returns i as soon as a match is found **M1**

(e) returns -1 after the loop completes with no match found; any correct logic, ignore minor syntax **A1**

(e) Answer:

```
FUNCTION LinearSearch(list : ARRAY[1:8] OF INTEGER, t
    FOR i <- 1 TO 8
        IF list[i] = target THEN
            RETURN i
        ENDIF
    NEXT i
    RETURN -1
ENDFUNCTION
```

Question 2

(a) pass 1: [3, 6, 1, 8], swaps = 3 **B1**

(a) pass 2: [3, 1, 6, 8], swaps = 1 **B1**

(a) pass 3: [1, 3, 6, 8], swaps = 1 **B1**

(a) pass 4: [1, 3, 6, 8], swaps = 0 (array already sorted) **B1**

(a) Answer: Sorted array: [1, 3, 6, 8]

(b) use a boolean flag (e.g. swapped) that is reset to false at the start of each pass, oe **B1**

(b) if no swaps occur during a full pass, the flag stays false, meaning the array is fully sorted, so the algorithm can stop early instead of running further unnecessary passes, oe **B1**

(b) Answer: Use a flag that records whether any swap occurred in a pass; if a whole pass completes with no swaps, the array is sorted and the algorithm can terminate early.

(c) outer loop controlled by a swapped flag, e.g. WHILE swapped = true **M1**

(c) swapped is reset to false at the start of each pass **M1**

(c) inner FOR loop comparing each pair of adjacent elements, e.g. FOR j <- 1 TO n - 1 **M1**

(c) correct swap of arr[j] and arr[j+1] using a temporary variable when arr[j] > arr[j+1] **M1**

(c) swapped is set to true whenever a swap occurs **M1**

(c) algorithm correctly terminates once a full pass makes no swaps; any correct logic, ignore minor syntax **A1**

(c) Answer:

```
PROCEDURE BubbleSort(BYREF arr : ARRAY[1:n] OF INTEGE
    DECLARE i, j, temp : INTEGER
    DECLARE swapped : BOOLEAN
    swapped <- true
    WHILE swapped = true
        swapped <- false
        FOR j <- 1 TO n - 1
            IF arr[j] > arr[j + 1] THEN
                temp <- arr[j]
                arr[j] <- arr[j + 1]
                arr[j + 1] <- temp
                swapped <- true
            ENDIF
        NEXT j
    ENDWHILE
ENDPROCEDURE
```

(d) $O(n^2)$ **B1**

(d) Answer: $O(n^2)$

© Revision Library | revisionlibrary.co.uk

Question 3

(a) Push (add data) **B1**

(a) Pop (remove data) **B1**

(a) Answer: Push and Pop

(b) Push(5): top=1, stack=[5] **B1**

(b) Push(9): top=2, stack=[5,9] **B1**

(b) Pop() returns 9: top=1, stack=[5] **B1**

(b) Push(3) then Push(7): top=3, stack=[5,3,7] **B1**

(b) Pop() returns 7: top=2, stack=[5,3] (final state) **B1**

(b) Answer: Final state: top = 2, stack contents [5, 3]

(c) a stack is LIFO (last in, first out), oe **B1**

(c) each opening bracket is pushed onto the stack, and each closing bracket should match (pop) the most recently pushed, still-unmatched opening bracket, oe **B1**

(c) if the stack is empty at the end (and never needed to pop from empty) the brackets are balanced; a queue's FIFO order would not correctly match the most recent unmatched bracket, oe **B1**

(c) Answer: A stack's LIFO behaviour naturally matches each closing bracket with the most recently opened, still-unclosed bracket; a FIFO queue would match brackets in the wrong order.

(d) any valid example, e.g. a print queue, a task/process scheduling queue, a keyboard input buffer **B1**

(d) Answer: Example: a print queue, where documents are printed in the order they were sent.

Question 4

(a) n factorial (n!), for positive integer n **B1**

(a) Answer: n factorial

(b) Mystery(4) calls Mystery(3), which calls Mystery(2), which calls Mystery(1), correctly shown in sequence **B1**

(b) Mystery(1) is the base case (n=1) and returns 1 **B1**

(b) Mystery(2) returns 2 * 1 = 2 **B1**

(b) Mystery(3) returns 3 * 2 = 6 **B1**

(b) Mystery(4) returns 4 * 6 = 24, cao **B1**

(b) Answer: Mystery(4) = 24

(c) the base case is n = 1, but starting from n = 0 and repeatedly subtracting 1 gives 0, -1, -2, ... which never exactly equals 1, oe **B1**

(c) so the base case is never reached and the function calls itself indefinitely (infinite recursion), oe **B1**

(c) each call uses memory on the call stack; because the recursion never stops, this is called a stack overflow **B1**

(c) Answer: Because n decreases past 1 without ever equalling it, the base case (n=1) is never reached, causing infinite recursion and a stack overflow.

(d) result initialised to 1 before the loop **M1**

(d) FOR loop iterating from 1 to n (e.g. FOR i <- 1 TO n) **M1**

(d) result <- result * i inside the loop **M1**

(d) correct RETURN result after the loop; any correct logic, ignore minor syntax **A1**

(d) Answer:

```
FUNCTION MysteryIterative(n : INTEGER) RETURNS INTEGE
    result <- 1
    FOR i <- 1 TO n
        result <- result * i
    NEXT i
    RETURN result
ENDFUNCTION
```


© Revision Library | revisionlibrary.co.uk